TECHNICAL REPORT 3061
February 2017

Risk Metrics for Android™ Devices

Roger A. Hallman
Megan Kline

SSC Pacific
San Diego, CA 92152-5001

SSC Pacific
San Diego, California 92152-5001

G. M. Bonitz, CAPT, USN
Commanding Officer

C. A. Keeney
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed by the Network Security Engineering Services and Operations Branch (Code 58230) of the Computer Network Defense-Cyber Security Division (Code 58200), Space and Naval Warfare Systems Center Pacific (SSC Pacific), San Diego, CA.

Released by
J. Romero-Mariona, Head
Network Security Engineering Services and
Operations Branch

Under authority of
J. Elissa, Head
Computer Network Defense-
Cyber Security Division

SB

EXECUTIVE SUMMARY

The Android™ Operating System (OS) is far and away the most popular OS for mobile devices. However, the ease of entry into the Android™ Marketplace allows for easy distribution of malware. This report surveys malware distribution methodologies, then describes current work being done to determine the risk that an "app" (application) is infected. App analysis methods discussed include code and behavioral analysis methods.

CONTENTS

1. INTRODUCTION

The Android™ Operating System (OS) is the most popular platform for mobile devices, currently holding at least 80% of marketshare [1], which continues to grow and is expected to reach more than 1.6 billion users by 2020 [2]. The platform offers an open entry point into the "app" (application) market for developers, which also gives malicious actors access to millions (potentially billions) of devices [3].

This report gives an exposition on risk metrics for applications in the Android™ OS ecosystem and is organized as follows. First, common malware distribution methods are explained and suggestions for mitigation are made. Then both static code analysis and behavior-based analysis methods for risk assessment are surveyed. Several other approaches to risk analysis, such as crowd sourcing risk analysis, are surveyed.

2. MALWARE AND ITS DISTRIBUTION

Malware may include trojans, backdoors, worms, botnets, spyware, adware, and ransomeware:

- Trojans appear to be benign apps, but perform harmful activities without user knowledge or consent. A recent trend in trojans has seen attacks on multi-factor authentication for banking transactions, capturing usernames and passwords and then stealing Mobile Transaction Authentication Numbers (mTANs) to silently complete transactions [4].
- Backdoors allow outsiders to stealthily enter the system, bypassing normal access features.
- Worms spread exact copies of themselves and spread through the network, for instance Bluetooth worms will copy themselves on paired devices [5].
- Botnet apps allow for a device, called a *bot* to be controlled by an outside server, and collections of bots to act in concert as *botnets*. Botnets are often used in Distributed Denial of Service (DDoS) attacks. Botnet apps may include instructions to download other malicious packages.
- Spyware often presents itself as a good utility, but monitors user activities in the background (e.g., contacts, messages, locations, etc.).
- Adware is used to target a device user for personalized advertisements, which may degrade operations.
- Ransomeware [6] locks a user's device, making it inaccessible until a ransome is paid through an online payment service (oftentimes using a cryptocurrency like Bitcoin, which makes payments untraceable [7]).

2.1 IS A POPULAR APP FREE? THE THREAT POSED BY REPACKAGING

Oftentimes, multiple versions of an app will appear in the app store when a potential user searches for them. One version of an app will have a monetary cost associated with it while other versions will be listed as free to the user. The version of the app with a monetary cost associated with it will generally be benign while the free version(s) will be repackaged to include adware [8] (not explicitly malicious) or malware [9] (explicitly malicious) inserted in the underlying code. There are reports that 77% of the top 50 free apps in the Google Play Store are repackaged [10]. Repackaging is among the most common methods for Android™ malware distribution [11, 12]; repackaging and risk can be minimized by ensuring that only the official paid-for app is downloaded.

2.2 OTHER DISTRIBUTION TECHNIQUES: DRIVE-BY DOWNLOADS, DYNAMIC PAYLOADS, AND STEALTH TECHNIQUES

In addition to repackaging of apps, malware may be distributed by other means.

2.2.1 Drive-by Downloads

A drive-by download (DbD) occurs when malware is unintentionally downloaded in the background [10]. The DbD attack has been a common tactic for malware distribution and propogation for many years [11]. Traditionally, the anatomy of a DbD attack would include an attacker compromising a legitimate Web application and uploading a malicious Javascript™. The ultimate victim of the DbD will become infected by sending a Web request to the server, which sends the malicious code along with the legitimately requested Web page and the Javascript™ downloads its malicious payloads [13]. Many of these attacks target

third-party browser plug-ins, which introduce "low-hanging fruit" vulnerabilities [13, 11], because they may undergo less testing. Many plug-ins are often written in languages like C, without concern for security best practices, and are distributed as executable binaries [13].

DbD attacks for mobile devices do not typically target browser vulnerabilities, but may entice users to download feature-rich apps. Zhou and Jiang [11] found four families of Android™ DbD malware:

- GGTracker uses in-app advertisements to attract users to malicious links. Users are usually subscribed to a premium rate service without their knowledge [14].
- Jifake is downloaded by malicious Quick Response (QR) codes that redirect the user to a URL containing it. The malware sends several SMS messages to a premium-rate number, and is the first known case of a malicious QR code-based attack [15].
- SpyEye-in-the-Mobile (Spitmo) targets online banking mobile transaction authentication numbers (mTANs) by directing users to download an app that promises safer online banking. Once downloaded, user mTANS and Short Message Service (SMS) messages are sent to a remote server [16].
- ZeuS-in-the-Mobile (Zitmo) is a variant of the ZeuS botnet and, like Spitmo, targets online banking activities, primarily in Europe. It steals mTANs and sends them to remote servers [17].

2.2.2 Stealth Malware Techniques

Malware developers are taking advantage of the limited resources available on Android™ devices that limit scanning capabilities. Hardware vulnerabilities and obfuscation of malicious code allow easy avoidance of anti-malware apps [18]. Some of these techniques include key permutation, native code execution, code encryption, and Java™ reflection [10].

2.2.3 Update Attacks

An Update Attack is a malware distribution technique that is very resistant to static scanning techniques. Instead of downloading the entire malicious package, there is only an update component that fetches the malicious payload at runtime [11]. Some examples of Update Attacks include:

- BaseBridge [19] includes one exploit, "rage against the cage", which downloads a payload application which then attempts a privilege escalation attack to gain root access. It transmits personally identifiable information to a remote server [20].
- DroidKungFuUpdate downloads malware through a third-party library that provides a legitimate notification functionality [11].
- AnserverBot [21] upgrades certain components within their host app. Since the entire app is not updated, user permission is not required. (This is a distinct difference from BaseBridge and DroidKungFuUpdate.) AnserverBot retrieves a public (but encrypted) blog entry containing the payloads for the update.
- Plankton [19] operates similar to AnserverBot, except it downloads additional dex files to dynamically extend its capabilities.

3. CODE ANALYSIS METHODS

An effective method for assessing risk of malware infection in Android™ apps is the comparison of "feature vector" analysis. Feature vectors summarize important characteristics of an object's state [22] and feature vector elements for a given app are characteristic actions taken by that app. Apps are classified into different categories based on their claims. Benign apps within each category that have similar functions are expected to have similar permissions requests, while malicious apps deviate. The extent of this deviation can be useful for assessing the risk of malware infection in an app. The use of weighting mechanisms [23] are useful for feature vector analysis.

Almost all risk assessment for Android™ apps are permissions-based. Reference [24] shows a feature vector analysis approach to to risk metrics by considering the vector of permissions requests. Apps are separated into 29 categories and then the permissions tendencies of benign apps are used to create baseline permissions vectors for each category. There are 144 permissions available in the Android™ operating system; however, 53 of these were not available to third-party applications, and so the remaining 91 permissions were used for a permissions vector. A 91-dimension Boolean vector was constructed with a 1 denoting when a permission was requested and a 0 when it was not. Every app is represented in this manner and compared against a category baseline permissions vector using a weighted Euclidean metric. In the Cartesian Coordinate System, $\mathbf{p} = (p_1, p_2, ..., p_n)$ and $\mathbf{q} = (q_1, q_2, ..., q_n)$ are points in a Euclidean n-space and the distance between them is defined by

$$(\mathbf{p}, \mathbf{q}) = d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=0}^{n} (q_i - p_i)^2}. \tag{1}$$

When certain coordinates are deemed of greater or lesser importance, a weighted Euclidean metric is used:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=0}^{n} w_i (q_i - p_i)^2}. \tag{2}$$

Specifically, 24 permissions that were seen as risky were given a weight of $w_i = 2$, while the others were simply given a standard weight of $w_i = 1$.

Two data sets were used for testing this methodology. Because the authors are Chinese, they chose to download apps from the Tencent Android™ Market, a leading Android™ market in that country. This data set consisted of 7099 verified benign apps in various categories. The second data set consisted of 1260 malware samples downloaded form the Android™ Malware Gene Project [25], covering the majority of Android™ malware families. The market data set of 7099 verified benign apps were mainly used to create baseline vectors for each app category.

Sanz et al. [26] use sets of feature vectors of permissions and uses-features for anomaly detection. This information is attained from the `AndroidManifest.xml` file that is packed with every Android™ app. Uses-features, while optional information, determine the feature used by the app.

A 130-dimension permissions vector and a 37-dimension uses-feature vector (where possible) were generated from this information. "Normality" (baseline) models were developed using the information in the `AndroidManifest.xml` files from 1,811 verified benign samples, while 2,808 malware-infected samples were tested for anomalies. Multiple metrics were used for anomaly detection and risk analysis, including the Euclidean metric (1), the Manhattan metric, and cosine similarity. The Manhattan metric is the

distance between two points p and q, summing the lengths of the projections of the line segments between the points onto the coordinate axes:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sum_{i=0}^{n} |\mathbf{p}_i - \mathbf{q}_i|. \tag{3}$$

Cosine Similarity measures the similarity between vectors by finding the cosine of the angle between them. Measuring distance and not similarity (and hence a metric) requires the use of $1 - CosineSimilarity$:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = 1 - \cos(\theta) = 1 - \frac{\mathbf{p} \cdot \mathbf{q}}{||\mathbf{p}|| \cdot ||\mathbf{q}||}. \tag{4}$$

Grace et al. [27] detail a prototype system, *RiskRanker*, which utilizes multi-order code analysis to determine the risk of malware in apps. RiskRanker is meant to be a proactive scheme that sifts through Android™ markets spotting zero-day malware without relying on malware specimines or their signatures.

Apps are analysed and separated by potential risk levels:

- High Risk apps exhibit platform-level software vulnerabilities that could be used to compromise device integrity without proper authorization.
- Medium Risk apps can cause financial loss or disclose private information about the user, but do not exploit software vulnerabilities.
- Low Risk apps may collect general personal or device-specific information.

RiskRanker subjects apps to two sets of code analysis. The First Order Analysis are designed to expose High and Medium Risk apps. High Risk Apps are detected by distilling known vulnerabilities into vulnerability signatures that capture their essential characteristics, which are exploited when the vulnerability is exploited. Apps are pre-processed to detect the presence of native code, and these apps are checked for the presence of root exploits. Medium Risk Apps are detected by making use of Android's well-defined conditions for callback invocation. This assumes that when a malware intends to charge the device user or transmit sensitive data without their knowledge, it is unlikely to ask permission via a callback handler. RiskRanker performs static analysis on Dalvik [28] bytecode contained in each app, using control- and data-flow analysis to unambiguously identify the callbacks that call to a method of interest.

Dalvik is a virtual machine that runs applications and code written in Java™. Dalvik requires that control-flow graphs be reliably determined in advance. Once code is loaded for execution, Dalvik employs a static verifier to ensure that methods and classes are well-formed and can be resolved. However, this verification is only for intra-method data-flow analysis, determining the type of virtual register at each point in the program [27]. Moreover, Android™ apps do not require being called in any strict sequence and this leads to apps having a very complicated lifecycle. The control-flow graph is traversed in reverse, searching for callback methods that do not imply user interaction. This approach of backwards slicing to determine where the arguments of a network call originate leads to spotting execution paths for potential information leaks.

The First Order Analysis excels at handling non-obfuscated apps, but may not be able to detect malware that employs encryption or dynamically changes its payload. RiskRanker's Second Order Analysis collects and correlates behaviors that are common among malware. The first step in the Second Order Analysis is capturing distinctive behavior that is not likely malicious in and of itself, but is abused by existing malware. An example of this type of behavior is the inclusion of a child app [29], within the host app, that can escalate privileges and even remain after the host has been uninstalled. Another example is the use of Java's encryption APIs (application program interfaces) [30] for encrypting their data and communication. These examples are commonly used by legitimate apps and can be abused by malware. Specifically, to recognize these behaviors, the following information is automatically collected:

1. Child app location,
2. Background dynamic code loading and execution paths,
3. Programmed access to `assets`/`res` directories,
4. Use of encryption and decryption methods, and
5. Native code execution and Java Native Interface (JNI) access.

Malware oftentimes stores exploit code in an encrypted format within the `assets` or `res` directories. This code will be decrypted and read at runtime. This behavior helps evade the code analysis in the First Order Analysis. The `assets` and `res` directories are designed to contain art assets, user interface descriptions, and so on, but may also contain arbitrary data. Accessing these directories in a code path that includes encryption and execution methods should be considered suspect, and it goes without saying that storing native binaries in such unusal circumstances suggests that an app is hiding something. Java™ provides native support for cryptography with the `javax.crypto` package. With easy to use, standardized encryption and hashing functions, this is an alluring package for malware developers (although in the future they may use difference cryptography packages), and so RiskRanker searches specifically for the use of its APIs. Encrypted native binaries will be run once they have been decrypted and loaded, so JNI calls and other invocations of native code are also noted.

Android™ provides a functionality that can be used to load and execute bytecode from an arbitrary source at runtime. More specifically, any app can make use of the `DexClassLoader` to load classes from embedded `.jar` and `.apk` files. This kind of unsafe dynamic loading of untrusted Dalvik code is noted in the Second Order Analysis phase of RiskRanker. None of the Second Order Analysis behaviors in and of themselves imply that an app is malware, and many legitimate apps make use of these functionalities. For example, dynamic loading is used by many legitimate apps for updating functionality without reinstalling the app itself.

RiskRanker was tested with 188,318 apps collected over several months, and successfully detected 718 malicious apps from 29 malware families, including 322 zero-days from 11 malware families. The authors point out that RiskRanker is not a panacea for several reasons:

- Some malware families do not engage in behavior that the system was designed to detect. For instance, malware that utilizes social engineering attacks are difficult for an automated system to detect.
- Malware may not share the same payload as other members of its family.
- Some apps are “guilty by association,” such as an installer that is not itself malicious but is used to install a malicious payload.

4. BEHAVIORAL ANALYSIS

As far back as 2011, Burguera et al. [31] developed a framework for Android™ malware risk analysis that crowdsourced app behavior by collecting traces of application execution traces. This framework relies on three main components: data acquisition, data manipulation, and a malware analysis and detection system. The data acquisition component collects application data from the user, including basic device information, a list of installed applications, and system call log files. The data manipulation component manages and parses all the data collected; device information is stored in a central database and system call traces are used to create a feature vector for analysis. Malware analysis and detection is done by using K-means clustering [32] over the crowd-sourced system call vectors to create a "normality model" and detect anomolous behavior.

A lightweight app, *Crowdroid*, was developed that is available for installation from Google's Market and monitors Linux® Kernel system calls, sending them to a centralized server. Non-personal, behavior-related data is sent to the server for analysis. Once parsed, a baseline feature vector of system calls is developed. System calls are how a program requests services from the OS's kernel and provide useful functions to applications like network-, file-, or process-related operations. The Linux® kernel is executed in the lowest layer of Android's architecture, so all requests made from upper layers must pass through the kernel using the system call interface before execution in the hardware. Specifically, Crowdroid uses a Linux® tool, *Strace*, to collect system calls to generate an output file of all Android™ application events. This file provides useful information for each system call executed by an app (e.g., count, opened and accessed files, execution time stamps).

Burguera's proposed framework and app, Crowdroid, underwent limited experimental testing. Self-written programs were used to provide a normality model and then the same programs were modified with self-written malware. The framework claimed a 100% success rate in detecting self-written malware.

A second round of experimental testing featured two publicly available specimens of malware:

1. PJApps, contained in the Steamy Windows app - Steamy Windows gives the smart phone screen the appearance of being covered with steam and lets the user wipe it off. The PJApps malware starts a background application and is programmed to perform multiple functions.

2. HongTautou trojan, contained within the Monkey Jump 2 app - Monkey Jump 2 is a game that is freely available through Google's Play Store, but is repackaged with the HongTautou trojan through multiple third-party sites. HongTautou sends information and browses the Internet.

The experimental testing consisted of 20 clients running the Crowdroid app, and up to 60 user interactions with each app for malware discovery. Specifically, there were 60 interactions with the self-written apps and malware, boasting a 100% detection rate. There were only eight interactions with the Steamy Windows and PJApps apps and 20 interactions with Monkey Jump 2 and the HongTautou trojan. Crowdroid showed 100%detection of PJApps, but only 85% detection of HongTautou. The reason for the lower success rate in detectign HongTautou is that it makes fewer sytem calls, only sending information and browsing the Internet.

5. OPTIMIZATION OF RISK ANALYSIS PROCESSES

Android™ app markets, both the official Google Play Store and alternative markets, simplify consumer software distribution. However, analysis of apps in these markets is a daunting task; the Google Play Store alone was approaching 2 million apps by the beginning of 2016 [33]. Chakradeo, et al., propose *Mobile Application Security Triage* (MAST) [34] an optimization of Android™ app market malware evaluation processes utilizing Multiple Correspondence Analysis (MCA) [35]. The triage process is analogous to the prioritization of patients in a hospital emergency room. Triage is neither diagnosis nor treatment, but allows resource allocation to go where there is the greatest obvious need while delaying or outright dismissing the treatment of others. MAST was developed to provide a rank-ordered list that can determine which apps in a market are suitable candidates for deeper, more costly analysis.

MCA is an extension of correspondence analysis that assists analysis of relationships between several categorically dependent vairables and is used with sets of observations described by a set of nominal variables [35]. MCA asks "individuals" a series of "questions" and maps each individual and answer to a set of coordinates in the "principle axes," where information is condensed so that the majority of information is reflected with just a few axes. These principle axes can also be indicators of "hidden variables" that give better insight into collections of answers—indeed, a skilled analyst can describe these hidden variables by the principle axes and resulting point clusters.

Given a point cloud where there are a number, N, of questions with answers describing individuals, a cloud with a large N will have certain dimensions that are more interesting than others. Moreover, the most interesting dimensions of a large cloud may not be known *a priori*. MCA transforms a cloud of N-dimensional coordinates into principal components, where the first dimension is guaranteed to show more information than the second, and the second more than the third, etc.

Two insights are used for transforming clouds into principal coordinates:

1. Scaling less common answers as more distinct than more common answers, and
2. Variance of data.

Mobile Application Security Triage (MAST) uses MCA to rank Android™ apps in order of relative suspicion. MAST works as follows:

1. App atributes that define interesting security properties are identified.
2. Related attribute sets are combined to create an MCA questionnaire.
3. MCA is run over multiple polls to generate rough indications of suspicious behaviors.
4. The rough indicators are merged to create a ranking of application suspiciousness.

Mobile Application Security Triage uses easy-to-obtain attribute information from the application manifest, without using market-specific metadata. Specifically, MAST considers four attributes:

1. Permissions restrict access to security-sensitive operations. MAST considers only the 114 permissions defined by the Android™ framework, though custom permissions could be considered in addition to these.

2. Intents provide interfaces between core platform functionality and interactions between third-party apps. Intent filters handle intents by specifying pre-agreed on action strings to customize user experiences. There are 92 action strings defined in the Android™ framework that MAST takes into account.

3. Native code allows third-party developers to include native libraries in their app. MAST only considers whether or not an app contains native libraries.

4. The .zip files have no restrictions on what they may hold, and MAST considers their presence or absence in an app's main archive.

REFERENCES

1. Xuan Lu, Xuanzhe Liu, Huoran Li, Tao Xie, Qiaozhu Mei, Dan Hao, Gang Huang, and Feng Feng. 2016. “Prada: Prioritizing Android Devices for Apps by Mining Large-scale Usage Data.” *Proceedings of the 38th International Conference on Software Engineering* (pp. 3–13). May 14–22, Austin, TX. ACM.

2. Michael Schirer, Ryan Reith, and Anthony Scarsella. 2016.“Smartphone Growth Expected to Drop to Single Digits in 2016, Led By China's Transition From Developing to Mature Market, According to IDC.” IDC Press release (March 3). Available online at http://www.idc.com/getdoc.jsp?containerId=prUS41061616. Accessed February 6, 2016.

3. Venkatesh Gauri Shankar and Gaurav Somani. 2016. “Anti-hijack: Runtime Detection of Malware Initiated Hijacking in Android.” *Procedia Computer Science 78*:587–594.

4. Victor Chebyshev and Roman Unuchek. 2014. “Mobile Malware Evolution: 2013.” *Kaspersky Lab ZAOs SecureList* (February 24). Available online at https://securelist.com/analysis/kaspersky-security-bulletin/58335/mobile-malware-evolution-2013/. Accessed February 6, 2016

5. Luca Carettoni, Claudio Merloni, and Stefano Zanero. 2007. “Studying Bluetooth Malware Propagation: The Blluebag Project.” *IEEE Security & Privacy 5*(2):17–25.

6. Monika Choudhary, Pavol Zavarsky, and Dale Lindskog. 2016. “*Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and Characterization*,” *Procedia Computer Science 94*:465–472.

7. Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. “Cutting the Gordian Knot: A Look under the Hood of Ransomware Attacks.” *Proceedingsof EURCOM. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 3–24). July 9–10, Milan, Italy. Springer.

8. Bartlomiej Uscilowski. 2013. “Mobile Adware and Malware Analysis.” *Security Response*. Symantec Corporation..

9. Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2015. “Android Security: A Survey of Issues, Malware Penetration, and Defenses,” *IEEE Communications Surveys & Tutorials 17*(2):998–1022.

10. Saba Arshad, Munam Ali Shah, Abid Khan, and Mansoor Ahmed. 2016. “Android Malware Detection & Protection: A Survey,” *International Journal of Advanced Computer Science and Applications 7*(2):463–475.

11. Yajin Zhou and Xuxian Jiang. 2012. “Dissecting Android malware: Characterization and Evolution.” *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (pp. 95–109). May 20–23, San Francisco, CA. IEEE.

12. Sajal Rastogi, Kriti Bhushan, and B. B. Gupta. 2016. “Measuring Android App Repackaging Prevalence Based on the Permissions of App,” *Procedia Technology 24*:1436–1444.

13. Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. 2009. “Defending Browsers Against Drive-by Downloads: Mitigating Heap-spraying Code Injection Attacks.” *Proceedings of the Sixth International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 88–106). July 9–10, Milan, Italy. Springer.

14. A Bibat. 2011. “Ggtracker Malware Hides as Android Market.” *Android Authority*. Available online at http://www.androidauthority.com/ggtracker-malware-hides-as-android-market-17281. Accessed February 7, 2017.

15. Xuxian Jiang and Yajin Zhou. 2013. “A Survey of Android Malware.” In *Springer Briefs in Computer Science:Android Malware*, pp. 3–20. Springer, New York, NY.

16. Matt Liebowitz. 2011. “New Spitmo Banking Trojan Attacks Android Users.” *TopSecurityNews Daily*. Available online at http://www.nbcnews.com/id/44509898/ns/technology_and_science-security/t/new-spitmo-banking-trojan-attacks-android-users/#.WJoKxWq7rcw. Accessed February 7, 2017.

17. Najla Etaher, George R. S. Weir, and Mamoun Alazab. 2015. “From Zeus to Zitmo: Trends in Banking Malware.” *Proceedings of the 9th IEEE International Conference on Big Data Science and Engineering, Volume 1* (pp. 1386–1391). August 20–22, Helsinki, Finland. IEEE.

18. Johannes Homann, Teemu Rytilahti, Davide Maiorca, Marcel Winandy, Giorgio Giacinto, and Thorsten Holz. 2016.”Evaluating Analysis Tools for Android Apps: Status Quo and Robustness Against Obfuscation.” *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (pp. 139–141). March 9–11, New Orleans, LA. ACM.

19. Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2014. “Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks,” *IEEE Transactions on Information Forensics and Security* *9*(1):99–108.

20. Yeongung Park, Chanhee Lee, Jonghwa Kim, Seong-Je Cho, and Jongmoo Choi. 2012. “An Android Security Extension to Protect Personal Information Against Illegal Accesses and Privilege Escalation Attacks,” *Journal of Internet Services and Information Security (JISIS)* *2*(3/4):29–42.

21. Yajin Zhou and Xuxian Jiang. 2011. “An Analysis of the Anserverbot Trojan.” NetQin Security Research Center, Raleigh, NC.

22. Dimitri P. Bertsekas and John N. Tsitsiklis. 1995. “Neuro-dynamic Programming: An Overview.” *Proceedings of the 34th IEEE Conference on Decision and Control, Volume 1* (pp. 560–564.). December 13–15, New Orleans, LA. IEEE.

23. Roberto Paredes and Enrique Vidal. 2006. “Learning Weighted Metrics to Minimize Nearest-neighbor Classification Error,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* *28*(7):1100–1110.

24. Huikang Hao, Zhoujun Li, and Haibo Yu. 2015. “An Effective Approach to Measuring and Assessing the Risk of Android Application.” *Proceedings of the 9th International Symposium on Theoretical Aspects of Software Engineering (TASE)* (pp. 31–38). September 12–15, Nanjing, Jiansu, China. IEEE.

25. Yajin Zhou and Xuxian Jiang. 2012. “Android Malware Genome Project.” Department of Computer Science, North Carolina State University, Raleigh, NC. Available online at http://www.malgenomeproject.org. Accessed February 7, 2017.

26. Borja Sanz, Igor Santos, Xabier Ugarte-Pedrero, Carlos Laorden, Javier Nieves, and Pablo G. Bringas. 2013. “Instance-based Anomaly Method for Android Malware Detection.” *Proceedings of the 10th International Conference on Security and Cryptography* (pp. 387–394). July 29–31, Reykjavik, Iceland.

27. Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. “Riskranker: Scalable and Accurate Zero-day Android Malware Detection.” *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (pp. 281–294). June 25–29, Ambleside, United Kingdom. ACM.

28. Najla Etaher and George R. S. Weir. “From Zeus to Zitmo: Trends in Banking Malware.” Technical Report 4:8. Department of Computer and Information Sciences. University of Strathclyde, Glasgow, Scotland..

29. Jonathan Stark. 2012. *Building Android Apps with HTML, CSS, and JavaScript*. O'Reilly Media, Inc. Sebastopol, CA.

30. Oracle. No date. “Java Cryptography Architecture (JCA) Reference Guide.” Oracle Java Documentation. Available online at https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#Introduction. Accessed February 7, 2017.

31. Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. “Crowdroid: Behavior-based Malware Detection System for Android.” *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 15–26). October 17–21, Chicago, IL. ACM.

32. Renato Cordeiro De Amorim and Boris Mirkin. 2012. “Minkowski Metric, Feature Weighting and Anomalous Cluster Initializing in k-means Clustering,” *Pattern Recognition* *45*(3):1061–1075.

33. Yuta Ishii, Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori. 2016. “Clone or Relative?: Understanding the Origins of Similar Android Apps.” *Proceedings of the 2016 ACM International Workshop on Security and Privacy Analytics* (pp. 25–32). March 9–11, New Orleans, LA. ACM.

34. Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. 2013. “Mast: triage for market-scale mobile malware analysis.” *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks* (pp. 13–24). April 17–18, Budapest, Hungary. ACM.

35. Hervé Abdi and Dominique Valentin. 2007. “Multiple Correspondence Analysis.” In *Encyclopedia of Measurement and Statistics*, pp. 651–657, Neil Salkind, Ed. Thousand Oaks, CA. Sage.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
February 2017	Final	

4. TITLE AND SUBTITLE

Risk Metrics for Android™ Devices

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

Roger Hallman
Megan Kline

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SSC Pacific
53560 Hull Street
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TR 3061

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SSC Pacific
53560 Hull Street
San Diego, CA 92152–5001

10. SPONSOR/MONITOR'S ACRONYM(S)

SSC Pacific

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Android™ Operating System (OS) is far and away the most popular OS for mobile devices. However, the ease of entry into the Android™ Marketplace allows for easy distribution of malware. This paper surveys malware distribution methodologies, and then describes current work being done to determine the risk that an app is infected. App analysis methods discussed include code and behavioral analysis methods.

15. SUBJECT TERMS

Android Operating System; botnet; bot; GGTracker; malware; drive-by downloads; trojans; backdoors; worms; adaware; ransomeware; stealth malware; Mobil Application Security Triage; Multiple Correspondence Analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Roger Hallman
U	U	U	U	20	**19B. TELEPHONE NUMBER** *(Include area code)* (619) 553-7905

INITIAL DISTRIBUTION

84300	Library	(1)
85300	Archive/Stock	(1)
58230	R. Hallman	(1)
58230	M. Kline	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (1)

Approved for public release.

SSC Pacific
San Diego, CA 92152-5001